HTTP adaptive streaming in mobile networks: characteristics and caching opportunities

Ali Gouta[1,2], Dohy Hong[3], Anne-Marie Kermarrec[2] and Yannick Lelouedec[1]

[1]Orange Labs,{ali.gouta, yannick.lelouedec}@orange.com
[2]INRIA Rennes Bretagne-Atlantique,{ali.gouta, Anne-Marie.Kermarrec}@inria.fr
[3]N2Nsoft,{dohy.hong}@n2nsoft.com

***Abstract*—Cellular networks have witnessed the emergence of the HTTP Adaptive Streaming (HAS) as a new video delivery method. In HAS, several qualities of the same videos are made available in the network so that clients can chose the best quality that fits their bandwidth capacity. This has particular implications on caching strategies with respect to the viewing patterns and the switching behavior between video qualities. In this paper we present analysis of a real HAS dataset collected in France and provided by the country's largest mobile phone operator. Firstly, we analyse the viewing patterns of HAS contents and the distribution of the encoding bitrates requested by mobile clients. Secondly, we give an in-depth analysis of the switching pattern between video bitrates during a video session and assess the implication on the caching efficiency. We also model this switching based on empirical observations. Finally, we propose WA-LRU a new caching algorithm tailored for HAS contents and compare it to the standard LRU. Our evaluations demonstrate that WA-LRU performs better and achieves its goals.**

Categories and Subjects Descriptors:
C.2.0 [COMPUTER-COMMUNICATION NETWORKS]: General

General Terms:
Measurement, Design

Keywords:
HTTP Adaptive Streaming, HTTP Live Streaming, Microsoft Smooth Streaming, Live, Catch-up TV, chunks, profiles, LRU.

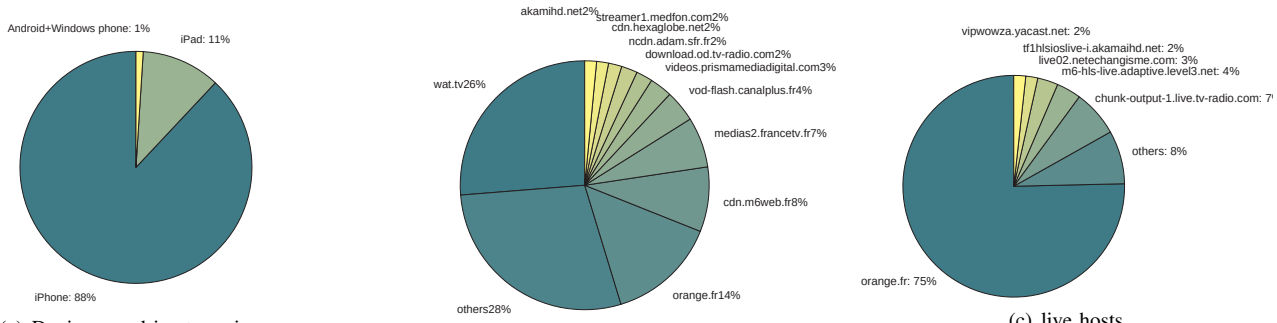
I. Introduction

Cellular networks have recently experienced a tremendous growth in data traffic. In [1], Cisco reported that mobile video traffic will reach 11.2 exabytes per month by 2017. This suggests that mobile streaming services will soon dominate the mobile communication landscape. Hence, ensuring a fast and a reliable content delivery accounting for users' network constraints is key to this evolution. HTTP Adaptive bitrate video Streaming (HAS) seems to solve this challenge since several encoding bitrates of the same content are provided to the mobile audience to sustain streaming experience and hence to achieve the highest quality of experience. In HAS, after generating several presentations of the same content, each presentation is segmented into smaller parts -called chunks or segments- and usually varies between 2 to 10 seconds length. When a user requests a video content, the hosting server sends back to the client a formalized description of the media presentation, named manifest file or Media Presentation Description (MPD), depicting all available bitrates of the requested content. The MPD enables the client-player to choose the quality that matches best his/her requirements, network and device capabilities.

HAS is widely used in mobile TV industry. Usually, TV broadcasters delegate to CDNs and to streaming servers the video segmentation process and to generate the adequate manifest file. CDN providers and Telcos are the most concerned to ensure a reliable and fast end to end delivery between users and servers. Therefore, studying HAS properties from an operator standpoint enables to accurately understand the clients' behavior and to provide guidelines to design adequate caching strategies tailored for HAS. As important as it is, this scope is not yet as well studied by the research community as others are (e.g. User Generated Content [2] or IPTV systems [3]). In this paper, we highlight the main features of HAS, in particular, we investigate:

- The behavior of mobile clients when requesting HAS contents for both live streaming and on-demand video streaming: Which laws best model the requested number of chunks during HAS sessions? How this behavior would be affected by radio constraints? What are the encoding profiles mostly requested by mobile clients? This clearly can be leveraged by caching mechanisms.
- The switching between video qualities during HAS sessions: How frequently mobile clients switch between the encoding bitrates during the session? How HAS affects the performance of proxy-caches? How can we characterize the switching pattern between bitrates (*i.e.* sojourn time per profile, start-up bitrate,...)
- We propose WA-LRU a novel caching mechanism that exploits the segmentation inherent to HAS and we validate the improvements throughout simulations.

The rest of the paper is organized as follows. Section II represents our dataset. In Section III, we study clients' behavior when requesting HAS streams. In Section IV we analyze the switching between video qualities and assess the implication of HAS on caching efficiency. In section V, we propose an effective caching algorithm that leverages the time-structure relationship of video chunks within a stream. In Section VI, we describe some related works and conclude the paper in Section VII.



(a) Devices used in streaming

(b) catch-up hosts

(c) live hosts

Fig. 1. End-to-end delivery: Devices and hosting servers for both catch-up contents and live channels

II. The Dataset overview

***Data collection**:* We have connected a server-log system at a Gi interface of one of the GGSNs deployed by the operator in France. We captured a total of 1.763.516 adaptive streaming session. We logged exactly a 92.595.115 HTTP GET request for each chunk requested. In our study, the considered GGSN serves a 230 NodeB/BTS. This area is covered by 3G/HSPA/HSPA+ and 2G (EDGE) radio access networks and served 246.913 unique active clients during the measurement period. The data collection was conducted over a period of 9 weeks from November 7^{th} 2012 until January 9^{th} 2013, involving mainly Apple HTTP Live Streaming (HLS) and Microsoft smooth streaming sessions (HSS). However, we limit our analysis on HLS sessions since they form more than 99% of HAS sessions in our dataset.

***Content types**:* HTTP adaptive streaming is mainly used by Over-The-Top TV companies. Telecom companies, on their side, offer their subscribers the TV-service as a free value-added service. Our dataset mainly contains:

- *Live TV sessions*, where clients watch live TV;
- *Catch-up TV sessions*, where TV content providers allow clients to replay a set of videos previously broadcasted in live. In our dataset, we observe that around 76% of HAS sessions corresponds to Live streaming sessions, the rest corresponding to catch-up TV.

Figure 1(a) shows that 99% of the HAS sessions are streamed using IOS based devices. IPhones' IOS versions are ranging from $V4.0$ to $V6.1$, IPads' IOS versions are ranging from $4.2.1$ to $6.0.2$. However, minority of sessions are streamed from Android and Windows phone-based devices.

Catch-up hosting servers: Figure 1(b) shows that the domain *wat.tv* is the most frequently requested domain name (25% of client requests) by mobile clients when requesting catch-up video contents. wat-tv is a TV service provider that hosts catch-up videos of most of the local French-TV broadcasters. Other TV channels delegates to world wide CDNs to deliver their contents on their behalf. For example: *"vod-flash.canalplus.fr"* CNAMES Akamai servers. Other TV companies also play the role of content service providers (CSPs) by using their own streaming infrastructure such as the domain *cdn.m6web.fr*.

Live streaming servers: Figure 1(c) shows that 75% of live HAS sessions are streamed from servers with domain name *orange.fr*: This means that subscribers usually access live TV channels through the Orange service. Orange provides streaming servers to enable local TV stations to broadcast their contents to mobile clients. The domain *chunk-output-1.live.tv-radio.com* is the second most requested domain (7% of client requests) which CNAMES Akamai servers to deliver live contents.

Nbr iof profiles	1	2	3	4	5	6	7	8
percentage	17%	22%	32%	13%	7%	4%	4%	1%

TABLE I. PERCENTAGE OF AVAILABLE PROFILES

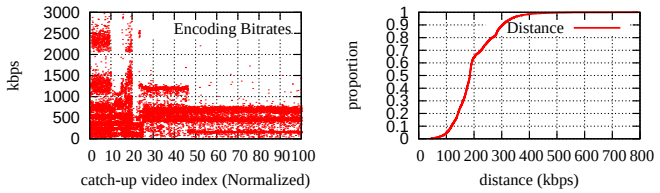
***Data processing**:* we identify a session by the first HTTP-GET request of the first chunk for which the URL address ends either with a *.ts* for a HLS stream, or with a *.ism* for a HSS stream. Each HAS session corresponds to the set of chunks sent to the associated client over a period of time. To perform our analysis, our logging system captures all packet headers of HAS streams which contain useful information, such as the packet size and sequence number. All information is aggregated per TCP connection and exported into a database, from where they are analyzed. Each persistent-TCP connection corresponds to one downloaded video segment. This means that the number of downloaded chunks during one session is equal to the number of persistent TCP connections established between the server side and the end-user over a period of time. The encoding schema of the different profiles may differ among video contents since each of the TV service providers may define his own range of encoding rates.

Figure 2(a) shows the distribution of the encoding profiles of all catch-up contents within the data-set. We observe that most of the defined encoding profiles are below 1000 kbps. In Table I, we show the proportion of number of the available profiles for catch-up TV contents requested during the measurement period. In Figure 2, we estimate the average distance in kbps between the encoding profiles defined for each catch-up video:

$$distance = \frac{\sum_{i>1}^{Nb\ profiles-1} (profile_{i+1}\text{-}profile_i)}{Nbr\ profiles}$$

We observe that in 95% of catch-up contents, the average difference between the encoding profiles is higher than 100

kbps. For this reason and since we are interested in aggregated statistics to draw conclusions about HAS characteristics, we define a scale of 8 different profiles (see Table II) that will be used to map each requested chunk from each HAS session to the appropriate profile *(P)*.



(a) encoding profiles (b) distance between profiles

Fig. 2. Profiles presentation used for catch-up contents

We assume that the encoding rate is equal to the volume of data contained in the packet payloads per one chunk and divided by the chunk's duration, which we get from the manifest file relative to each session. The measurement of the encoding rate of each chunk is initiated the time we capture the HTTP-200 response from the hosting server. When the download ends, we map the encoding rate to the corresponding profile.

Profile i	Encoding bitrate (kbps)
Profile 0 (P_0)	< 50
Profile 1 (P_1)	[50-150[
Profile 2 (P_2)	[150-280[
Profile 3 (P_3)	[280-420[
Profile 4 (P_4)	[420-600[
Profile 5 (P_5)	[600-1000[
Profile 6 (P_6)	[1000-2000[
Profile 7 (P_7)	$\geq$ 2000

TABLE II. PROFILES

Fields description: We have extended our previous work [2] (to be published soon) to give a particular interest to the switching between bitrates on large scale, as well to study time specificity of the transitions. To achieve this level of granularity, we proceed as follows: In each bitrate-switching we create a new log entry, in which we record the timestamp relative to this transition and the newly visited profile (P_i). The following data model shows in a tree hierarchy the information contained in each log-entry:

$Session_{id}$
Session Start (in timestamp).
URL
Cumulative number of requested chunks in session *id*.
Current profile $P_{j\epsilon[0..7]}$.
Next profile $P_{k\epsilon[0..7],k\neq j}$ (when a bitrate-switching happens).
Cumulative number of requested chunks within profile $P_{j\epsilon[0..7]}$.
Cumulative Bytes downloaded in profile $P_{j\epsilon[0..7]}$.
Cumulative time to deliver all requested chunks in profile $P_{j\epsilon[0..7]}$.

III. ANALYSIS OF CLIENTS' BEHAVIOR

A. Distribution of requested chunks per session

We are now interested to analyze the viewing pattern of mobile clients. In HAS, video contents are segmented in chunks and requested independently therefore this gives the opportunity to study the behavior of clients at chunk level. Figure 4 shows the distribution of the number of requested chunks per catch-up TV and Live HAS sessions.

Using the maximum likelihood (MLE) estimation, we determined that the Log-normal (μ, σ_1) distribution is best to match the requested number of chunks for both live and catch-up TV video sessions for the first 40 chunks per session. Log-normal distribution is ubiquitous to describe several mobile communication patterns such as the call holding times [3], the file data transfer in the mobile networks [4]... It once again fits well the behavior of clients when streaming HTTP adaptive contents for the first 40 chunks per session. Then, the generalized pareto distribution (GPD (k, σ_2, θ)) best models the tail observed when the number of chunks exceeds 40 chunks per session. Pareto distribution is well known to model *long-tails*, commonly used to describe the non-popular video contents [5]. In our case this indicates that few sessions request advanced chunks position. We show in Table III, the parameters of both log-normal and GPD distributions used to model the requested number of chunks per session. This is goes beyond the characterization that we made in [2] where we assumed that log-normal law may simply model the requested number of chunks. However, the *long-tail* needs a sharper distribution which is best modeled by pareto law. Combining Log-noraml and GPD, we may find a fine-grain model to fit the real-world access pattern of video chunks.

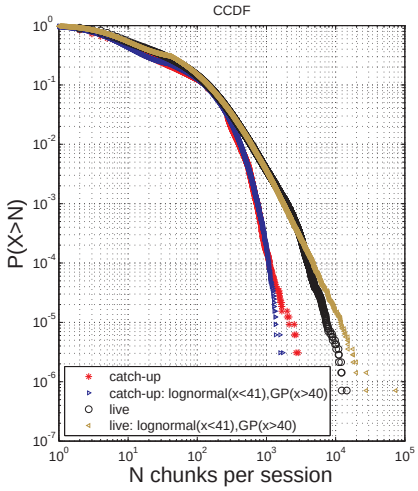
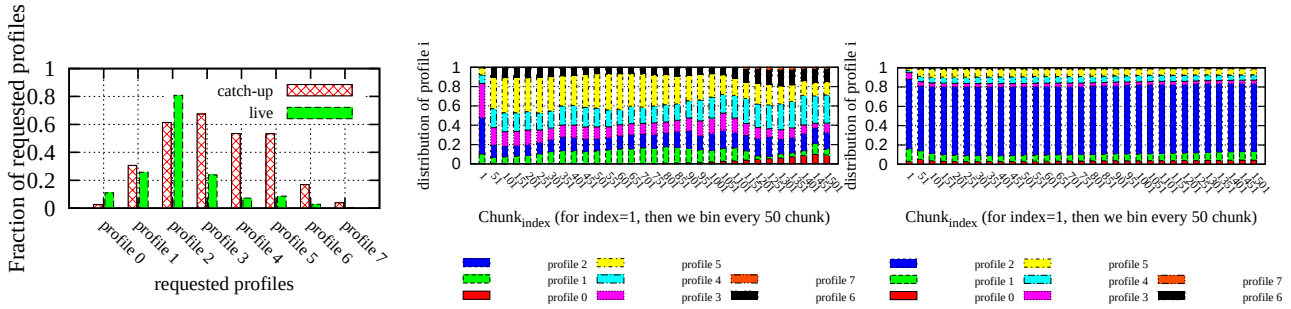


Fig. 4. Distribution of number of chunks per session for live and catch-up HAS sessions

	Log-normal ($x \leq 40$)		Generalized pareto ($x > 40$)		
params	μ	σ_1	k	σ_2	θ
catch-up	1.75145	1.01198	0.034218	118.55	40
live	1.87679	1.00993	0.412856	78.5267	40

TABLE III. EMPIRICAL MODELS

In the case where the chunks are delivered by caches or edge servers in the case of CDNs, such a tailed profile would degrade the caching efficiency especially in the case of a limited storage capacity and the use of a reactive replacement strategy. From the cache standpoint, requesting such advanced chunks position within a stream would compete the most requested chunks within the cache although few of clients will



(a) Proportion of sessions requesting $profile_i$ (b) catch-up (c) live

Fig. 3. Distribution of requested profiles with respect to the $chunk_{index}$ for both catch-up and live video streaming

effectively reach that stage of video streams. This becomes more critical when the cache adopts the LRU (Least recently used) algorithm, where these chunks will be top-ranked the time they will be requested, thus pushing down chunks that will probably get requested in the near future. In the last section of this paper, we estimate the potential benefit of not caching the chunks forming the tail when deploying a proxy-cache between the clients and the hosting server.

B. Profiles in catch-up and live sessions

A key advantage and characteristic of HTTP Adaptive Streaming is the possibility for the users to dynamically change the encoding bitrate of the video content as a function of the state of the network. Figure 3(a) shows the fraction of sessions in which $profile_i$ (P_i) has been requested. First, we note that the sum of the fractions of the requests as per the 8 profiles is higher than one (both in the cases of the live TV HAS sessions and catch-up TV sessions), since clients may switch between profiles during the same session. We also observe that profile 3 was requested in 63% of catch-up sessions, followed by profile 2 (60%), followed by profiles 4 and 5 (52% of catch-up sessions). However, we observe a significant difference between the live and catch-up TV HAS sessions. We observe that Profile 2 figures in more than 80% of live HAS sessions. The rest of profiles are less requested in live TV HAS sessions. This calls for deeper investigations on how to best manage the specificities of live TV HAS sessions in a mobile context, in the content preparation process (at the time the chunks are generated and the manifest files being updated dynamically) as well as in the delivery process either for Live or catch-up sessions, for example with specific caching strategies contents and for the most frequently visited profiles.

In Figure 3(b) and 3(c), we bin $chunk_{index}$ every 50 chunk over all HAS sessions, where the $index$ ranges from 1 to the last requested chunk in the session and points to the position of the $chunk$ within the stream. We show in these figures the proportion of requested profiles at each binned $chunk_{index}$. Interestingly, In Figure 3(b), We observe that although profile 3 figures in 63% of catch-up sessions (as shown in Figure 3(a)), we find that its proportion is around 30% for the first requested chunk, then it consistently falls below 20% for $chunk_{index>50}$. This will be further investigated in the next section. We also observe that when requesting the first chunk of catch-up contents, clients mostly select lower profiles. We observe that for $chunk_1$, 82% of catch-up sessions, clients request profiles lower than profile 4. This is explained such as, the MPDs are configured in a way to make clients start always buffering lower video bitrates to reduce at best the delays required at the *joining-phase* (loading time). Then, they rather decide to switch to higher profiles so far they experience a high available bandwidth. This is well illustrated in Figure 3(b), since profiles higher than P_3 accounts for more than 60% of clients' requests when $chunk_{index\geq 51}$. However, we observe in Figure 3(c) that for live sessions, profile 2 is uniformly and mostly selected over all $chunk_{index}$.

C. Time constraints and clients' behavior

In HAS, client-players try to adapt at best the playback quality to match the available bandwidth. Authors in [6] studied throughout experiments the behavior of the buffer under bandwidth restrictions using open source tools. However, in real world and in the context of mobile communication, radio conditions are potentially the main reason that pushes clients to decide on the bitrate to request.

Figure 5 shows that the profile selection is potentially correlated to the Download Throughout (DT) experienced by the clients. In Figure 5, we bin DT every 500 kbps, we observe that -when DT is equal to 500 kbps- 82% of requests are to download profiles equal or below P_3. However the distribution of profiles equal or higher than P_4, is superior to 60% when clients experience $DT > 2Mbps$.

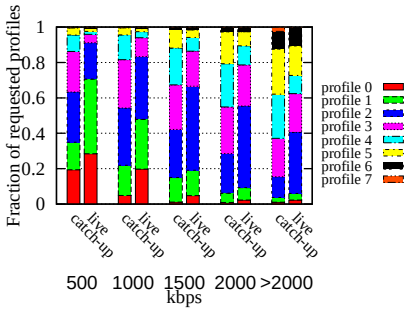


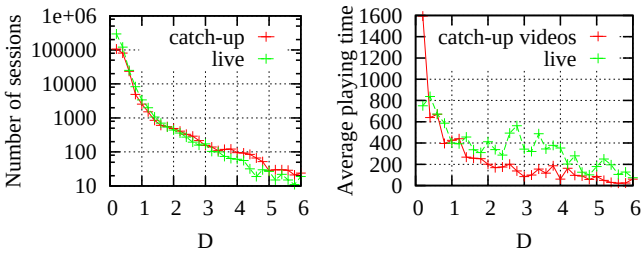
Fig. 5. Download throughput and Profiles interdependencies

So far, we study the *user-engagement* [7] by correlating the average playing time across all sessions as a function of the experienced average delay while delivering video chunks. The playing time in our case corresponds to the length of data that was downloaded during a session (*i.e. nb of requested chunks per session*Chunk_duration*). To do so, we define the delay metric (D) measured for each session,

which captures the average delivery time of one chunk per session divided by the average chunk duration per session:

$$D = \frac{\tilde{T}_{chunk}}{Chunk\,duration}$$

If $(D > 1)$, then the playback was potentially interrupted at least one time during the stream, as if the time spent in the network to deliver one chunk exceeds the chunk duration, this would certainly turn the client-player into the buffering mode. In Figure 6(b), we bin D in units of 0.2, and evaluate the playing time as a function of D. We stopped at $D = 6$, since as shown in Figure 6(a), the number of sessions in which D is superior to 6 is handful and thus not enough representative. For catch-up TV contents, Figure 6(b) shows that the average playing time is exponentially distributed as a function of D. When $D = 1$, we observe that the user-engagement decreases with a ratio of $\frac{3}{4}$, and still decreases as far as D becomes higher. However, surprisingly, for live sessions, we observe that the average playing time is still higher for $D > 1$ and does not show a clear pattern as the case for catch-up sessions. One possible reason is that when $D > 1$ clients might let their live session open while doing something else and waiting until the stream recovers.



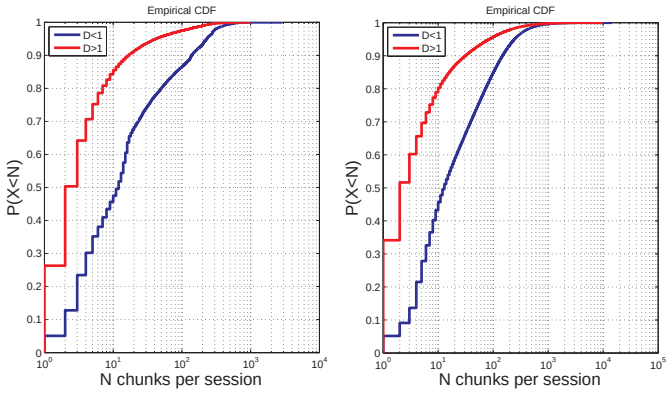
(a) Number of sessions = f(D) (b) user-engagement = f(D)

Fig. 6. Impact of the Delays in chunks delivery on the playing-time

We show in Figure 7(a) that when $D < 1$, clients are more likely to maintain their catch-up sessions active. We find that in 13% of catch-up sessions clients request at least 100 chunks. However, clients spend less time viewing a video when $D > 1$, mainly due to the video interruptions. We find that only in 2% of HAS sessions, clients request more than 100 chunks. We also observe that 50% of HAS sessions are aborted after requesting only 3 chunks per session when $D > 1$. This indicates that while experiencing long delays in the *joining-phase* (*i.e. D>1)*, we observe that one out of two sessions is aborted from the beginning. While delivering video chunks, it is worth to give priority to the first chunks of video contents to be served as fast as possible to guarantee a smooth *joining-phase* to the end-users. Our algorithm WA-LRU -which we represent in the last section- takes up this challenge since it gives priority to the first chunks to be cached. For live sessions (Figure 7(b)), we observe that in 15% of live sessions, clients request more than 100 chunks per session when $D < 1$, while only 3% of live sessions reach 100 chunks per session when $D > 1$.

IV. Switching between profiles

From now on, we concentrate only on catch-up sessions, since we observe a wide variety of requested profiles in catch-up sessions. However, this is not true for live sessions where clients tend to request one particular profile which is P_2. Hence, this makes the studies on the switching between bitrates and implications on caching-efficiency less significant in the live case, since clients do not adapt their bitrates as frequently as we observe for catch-up sessions.



(a) catch-up (b) live

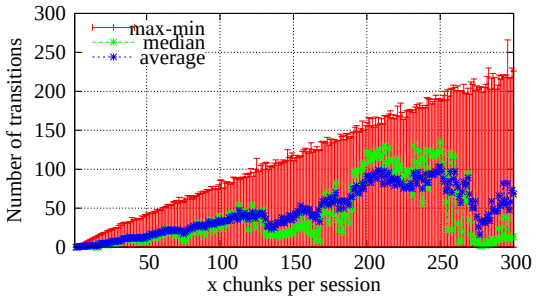
Fig. 7. CDFs of number of chunks per session when $D < 1$ and $D > 1$

A. Switching behavior

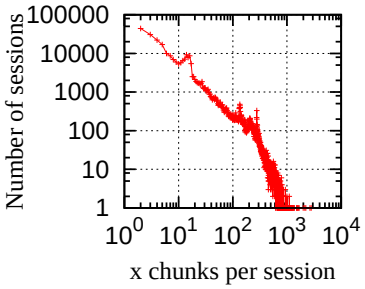
We show in Figure 8(a), the maximum, minimum, average and median number of transitions when requesting N chunks per session. We stopped at 300 chunks per session since minority of sessions reach effectively that stage of catch-up sessions (around 25 catch-up session, see Figure 8(b)), thus we assume that it is not enough representative. Figure 8(a) shows that whatever the number of requested chunks per session, the minimum number of transitions during a session is always 0. This indicates that in these sessions, clients never switched between bitrates either because the content provider might have defined a single video bitrate for the considered catch-up content, or clients may experience a high available bandwidth that prevented them from making any transition during the session. In the same figure, we also observe that clients start making transitions after requesting at least 2 chunks. This is a property of HLS where clients start buffering lower profiles before moving to higher ones in order to guarantee a smoother loading time. We also observe that in average, the number of transitions during a HAS session is bounded between $[1/6, 1/2]$ of the total requested chunks per session which is considered important and may adversely affect the end-to-end quality of service. We believe that switching between bitrates might be an efficient solution to sustain the video stream by allowing the clients suffering from resources scarcity to switch to lower profiles. However, assuming that we have a cache-server as a middle-box between clients and content-hosting servers, we show in the following that switching from one quality to another may affect the performance of the cache.

B. Impact of HAS on caching performance

To assess the implication of the switching between profiles on caching efficiency, we conduct a 15-days trace-driven simulation from the data-set and compare the case where clients are served with a *single* video quality (*e.g. such in*



(a) Number of transitions during HAS sessions



(b) Number of sessions reaching $chunk_i$

Fig. 8. Switching during HAS sessions

progressive download) to the case where clients are served with *multi-profiles* (*e.g.* such in HAS). To do so, we assume we have a cache deployed after the GGSN (*i.e.* on the same link where we have fixed the logging system).

For the sake of simplicity, we assume that:

- all chunks are 10 seconds length.
- we only consider the Catch-up TV sessions, since in Live sessions, chunks are generated on the fly. Hence, this introduces some complexity on assessing the time between generating the chunk on the fly, caching it and the difference in time between clients watching the same content in live.
- When we use *Multi-profiles* of the catch-up contents, we consider the original dataset and format it with respect to the following encoding profiles: $P_{i\epsilon[0..7]} = [40, 64, 240, 360, 440, 640, 1840, 2540]_{kbps}$.
- When we use *Single-profile* for all catch-up contents: we consider that all chunks are encoded at $640kbps$ (*i.e.* P_5).
- By default we use LRU as a cache replacement algorithm since it is largely adopted by CDNs, and scales well in large networks.
- Clients do not make any jump forward/backward during the catch-up session [1].

We evaluate the performance of the cache for different values of the capacity C such as: $C = \Gamma * S$, where S

[1]Previous studies on video streaming in mobile context show that up to 80% video sessions are without any trick mode (no pause and no jump forward/backward) [8], [9]

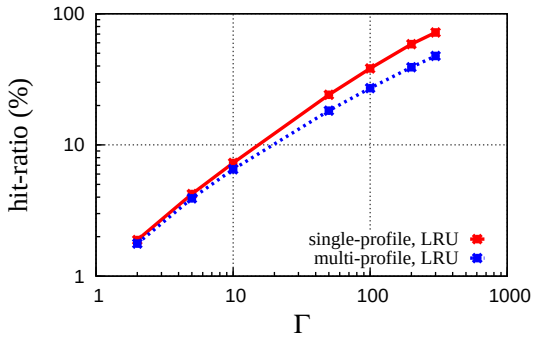


Fig. 9. Single profile VS Multi-profile: Implication on caching efficiency

represents a duration of 200 seconds (20 chunks), typically a short video-clip. In Figure 9 we assess the hit-ratio when using both single and multi profiles streaming technologies. When $\Gamma \geq 100$, we observe that although we have set the single-profile at $640kbps$ which is relatively high regarding the distribution of the requested profiles, we find that using a *single-profile* to encode video contents would increase the cache hit-ratio and may reach up to 15% of difference with regard to the *multi-profile* case. This is because some of the chunks belonging to the same video content are requested in different video qualities by users, and the cache would see them as different objects, since they have different URLs.

In most of the related works [10] [11], authors tried to propose several heuristics to estimate the bandwidth in advance and to select the appropriate profile. Authors tend to propose sensitive heuristics, and therefore clients will be so likely to change their video quality and this is without considering the implication on the end-to-end delivery chain. Raising the number of different qualities of the same $chunk_{index}$ from the same video content in the network would decrease the probability to find this chunk within the cache. This is well illustrated in Figure 9, where using *multi-profiles* of the same video content reduces the hit-ratio as comparing to streaming with a *single-profile*. As a consequence, clients would be redirected to the origin server, and hence further delays would be expected while downloading the video segments, and so far this would push client-players -that already implements a reactive adaptive streaming solutions- to trigger more and more the adaptation mechanism. To solve that, one possible solution would be to decrease the number of transitions (*i.e.* maximizing the sojourn time per profile) during the video session, while avoiding the buffer depletion and ensuring a high *user-engagement*.

C. Markov characterization of the switching between profiles

In the following we study the switching properties between profiles. To do so, we use Markov chains since they best describe and model this switching pattern. More precisely we use a continuous-time markov chain (CTMC), as the switching between bitrates is time-dependant during a HAS session (clients remain for a while in one particular profile before moving for another one), and as we only need to have a knowledge about the current profile to decide to which profile we should move next (no history is needed to be kept to decide to which profile we should move). In this part, we use our

data set to capture *empirical* distributions to characterize the CTMC. More precisely, we investigate:

- The sojourn time per $P_{i\in[0..7]}$
- The initial distribution $\pi(0) = [\pi_i(0)]$
- The infinitesimal generator
- State of the CTMC $(\pi(t))$ at instant $t > 0$

We start to formulate the switching process between profiles, and then identify step-by-step, all parameters needed to feed that model based on empirical observations. Formally, let t_0 be the time where a client decides to switch from profile P_i to $P_{j\neq i}$ in time slot dt, after spending an amount of time T_i in P_i. We note $p_{i,j}$ the probability to move from P_i to profile P_j.Then we have:

$$\begin{aligned} P_{\{i,j\}}(dt) &= P[(X(t_0+dt) = j | X(t_0) = i] \\ &= P[Ti \leqslant dt] * p_{i,j} \end{aligned} \quad (1)$$

Sojourn time per profile: $\mu_{i\epsilon[0..7]}$: $P[Ti \leqslant dt]$ represents the cumulative distribution when the client sojourn in P_i a duration $T_i < dt$. We use MLE estimation to evaluate the law that best matches the sojourn time in P_i. We find that the exponential distribution fits the empirical data well. We show in Table IV, the parameters per each profile.

$Profile_i$	exponential μ_i (seconds)
P_0	583.898
P_1	865.413
P_2	934.798
P_3	995.341
P_4	1107.89
P_5	1088.39
P_6	1128.9
P_7	1236.49

TABLE IV. SOJOURN TIME DISTRIBUTION

Then using Taylor series ($dt- > 0$), Eq (1) becomes:

$$\begin{aligned} P[Ti \leqslant dt] &= (1 - e^{-\mu_i * dt}) * p_{i,j} \\ &= \mu_i * dt * p_{i,j} + o(dt) \end{aligned}$$

This indicates that the *transition rate* $(\mu_{i,j})$ from P_i to P_j is also exponentially distributed, such as: $\mu_{i,j} = \mu_i * p_{i,j}$

The initial distribution $\pi(0) = [\pi_i(0)]$*:* In HLS, The client begins by retrieving the master MPD file which depicts the list of encoding profiles proposed by the content publisher. The media player begins playback with the first bitrate listed in the master MPD; it is expected that the first bitrate to be selected is the suggested bitrate of the content publisher [12]. Hence, all clients should start with a preset profile. This is in adequacy with what we observe in Figure 10. In this figure we show the percentage of videos in which clients request the same $chunk_{index}$ from the same catch-up videos, but at *N different bitrates*. We find that for $chunk_{index=1}$, all clients requesting the same catch-up video, start always by the preset profile in the manifest file (suggested by the content publisher). Then client-players change the video-bitrate as a function of the experienced bandwidth. We observe that for the range $chunk_{index\in[50..300]}$ (around 99% of HAS sessions, see Figure 4), 50% of video contents are requested with different profiles by mobile clients. Based on this, we assess the initial distribution $\pi(0) = [pi(0)_i]_{i\in[0..7]}$ of the first requested chunk $chunk_1$ over all HAS sessions, as follows:

$$\pi(0) = (0.0029 \quad 0.0981 \quad 0.3704 \quad 0.3473 \quad 0.0888 \quad 0.0843 \quad 0.0082 \quad 0)$$

$\pi(0)$ shows that in around 81% of sessions, clients start always requesting profiles lower than or equal to P_3, since clients start always requesting lower profiles to reduce delays at the *joining-phase*, then as Figure 3(b) shows, the distribution of requested profiles for $chunk_{index>50}$ is mostly concentrated between profiles 3,4 and 5.

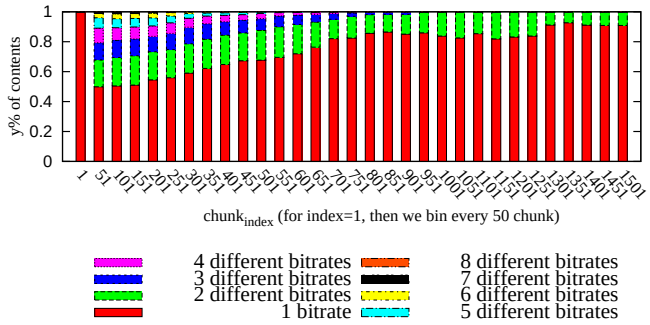


Fig. 10. Percentage of contents in which clients request different profiles when requesting the same $chunk_{index}$

The infinitesimal generator: Transition from $i- > j$*:* Let $Q = [q_{i,j}]$ be the infinitesimal generator of the CTMC we are characterizing, such as:

$$q_{\{i,j\}} = \begin{cases} \mu_{i,j} = \mu_i * p_{i,j};\ \forall\, i \neq j \\ -\sum_{j=0;j\neq 7}^{j=7} \mu_{i,j} = -\sum_{j=0;j\neq 7}^{j=7} \mu_i * p_{i,j};\ i = j \end{cases}$$

$(p_{i,j})$ are the coefficients of the transition probability matrix P from one state i (rows) to the next state j (columns) (i.e. The moment when the client-player decides to update his current bitrate from P_i to P_j). Using our dataset we assess P as follows:

$$P = \begin{pmatrix} 0 & 0.0997 & 0.2650 & 0.2685 & 0.1784 & 0.1358 & 0.0460 & 0.0066 \\ 0.0053 & 0 & 0.1533 & 0.3482 & 0.2683 & 0.1753 & 0.0464 & 0.0029 \\ 0.0036 & 0.0405 & 0 & 0.4237 & 0.28 & 0.2034 & 0.0456 & 0.0028 \\ 0.0021 & 0.0410 & 0.2496 & 0 & 0.4204 & 0.2486 & 0.0339 & 0.0041 \\ 0.0007 & 0.0192 & 0.1271 & 0.3867 & 0 & 0.4080 & 0.0500 & 0.0079 \\ 0.0009 & 0.0099 & 0.0817 & 0.2005 & 0.4790 & 0 & 0.2016 & 0.0260 \\ 0.001 & 0.0055 & 0.0303 & 0.0739 & 0.1617 & 0.6060 & 0 & 0.1213 \\ 0.0004 & 0.0018 & 0.0078 & 0.0160 & 0.0441 & 0.2266 & 0.7031 & 0 \end{pmatrix}$$

Hence we may infer the state of the CMTC at instant t. Let $\pi(t)$ be the distribution of $P_{i\in[0..7]}$ at $t > 0$. Since the time spent within a profile is exponentially distributed, then we have:

$$\pi(t) = \pi(0) * e^{Q*t}$$

$\pi(t)$ seems to achieve the stationary regime, since intuitively from figure 3(b), we observe that when progressing in the video stream, the distribution of $P_{i\in[0..7]}$ seems to be uniform for all $chunk_{index>50}$; except the case when the $index$ is superior to 900, which represents a handful number of sessions (see figure 8(b)).

V. CACHING HAS CONTENTS

Now we leverage the observation made in section III-A, and propose a novel caching mechanism -namely WA-LRU (Workload Aware-LRU)- that gives less priority to chunks with high indexes to be cached since they are less requested than the first ones.

A. Presentation of WA-LRU

WA-LRU leverages the segmented nature of HAS contents and cope with the long-tail caused by video sessions that reach advanced chunks position. Indeed, chunks forming the long-tail are subject for cache-eviction when using the native LRU, since it blindly caches the chunks objects without considering any property of the content (*i.e.* popularity, temporal locality of video chunks within catch-up contents,..). Upon a cache-miss, the cache will automatically pull the chunk from the parent-server or origin server, and thus evicts the least recently requested chunks in the LRU-list: typically chunks forming the tail or chunks that belong to non-popular contents. WA-LRU does not consider the popularity criteria of video contents since it is common that clients tend to request mostly popular contents. The challenge raised by WA-LRU is to decide whether to cache or not the recently requested chunk. This decision is made with regard to the *index* of the chunk within the video stream. The most challenging task -that WA-LRU address- is: how can we map the information we get from the instantaneous workload on the cache into the decision of caching or not the requested chunk. We define the workload -in bytes- on the cache over a period of T seconds as follows:

$$[W]_T = \int_t^{t+T} \textit{encode-rate(t)} * \textit{chunk-length(t)}\, \mathrm{d}t \qquad (2)$$

In Figure 12, we picked one representative week data from our data set and we plot the workload at a granularity of one hour (T=3600s). We observe that the workload follows a strong diurnal pattern. This let us suppose that our caching mechanism *should* consider the variation of the workload along the day (*i.e.* It should be aggressive at peak workload, and smooth at low workload).

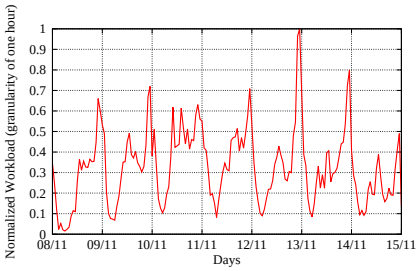


Fig. 12. Workload pattern measured by the logging system from 08/11/2012 to 15/11/2012

Under a high workload, a cache -running LRU- would be so reactive and would evict intensively chunk objects. We define the parameter R (Reactiveness of the cache) to assess the bytes' ratio to be evicted periodically at every T seconds and C represents the cache capacity.

$$R_T = \frac{[\text{Bytes evicted}]_T}{C} = \frac{[W]_T - [\text{Bytes served from the cache}]_T}{C} \qquad (3)$$

R_T is sensitive to the diurnal pattern since it depends on the workload. Under a high workload, R_T would increase and reach its maximum at peak hours. To the end of the paper, we set $T = 10seconds$ to match it to the chunk length. By doing so and if we assume that $R_{T=10s}$ is static or varies slightly at each period $T = 10s$ across the time (we further show that this assumption could be retained), then if a particular chunk is being requested at time t_0 (This will promote it to the top of the LRU ranking list), then it will be evicted from the cache if it would not be requested again for the next $t_0 + 10 * floor(\frac{1}{R_T})$ seconds. For the rest of the paper, we note $\Delta_T = floor(\frac{1}{R_T})$.

To illustrate our purpose, if we suppose that $floor(\frac{1}{R_{T=10s}}) = 500$, and that there is only one client requesting video k, and he is currently requesting the 501^{st} chunk. Therefore, assuming that clients do not use any trick-mode (jump backward or forward) while watching the content, then, it would be useless to cache this chunk, since we are sure in advance that no-client would request that chunk at least for the next $10 * \Delta_{T=10s} = 5000seconds$ (before being evicted). Hence, for each chunk which index is superior to 500, the cache will not cache it.

Since the value of $\Delta_{T=10s}$ depends on the workload, hence it will change across the time (*i.e.* becomes smaller at high workload, and bigger at low workload). We then consider the following weighted moving average method to compute the value of the threshold periodically at each *T=T+10s* and to which we compare each chunk's index as follows:

$$\begin{cases} Th_0 = \frac{1}{R_{T=10s}} \\ Th_T = (1-\beta) * Th_{T-10} + \beta * \frac{1}{R_{T=10s}} \end{cases}$$

If Th is lower than the $index$ of the chunk, then this chunk would not be cached; else it would be cached. β is used to avoid the instantaneous dramatic variation of $R_{T=10s}$. However, in our data set we find that the coefficient of variance of the workload ($Cov = \frac{\sigma}{\mu}$) over the day is equal to 0.7 which is inferior to 1. This indicates that the variation of R is smooth across the time. Based on this information and for the sake of simplicity we set $\beta = 1$.

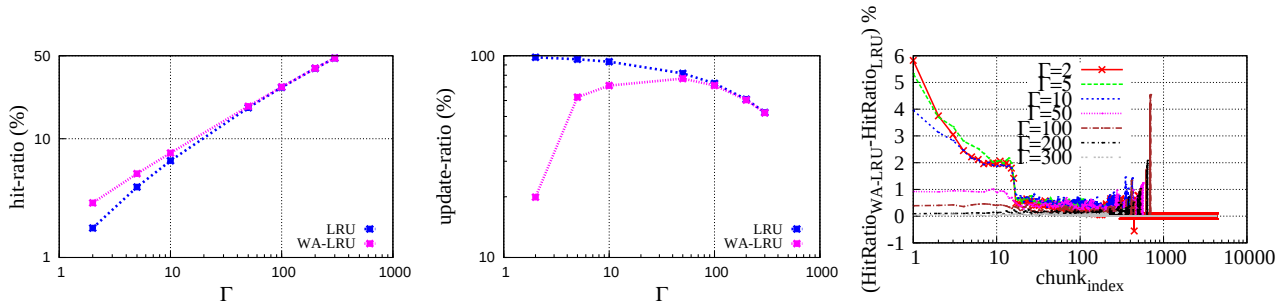
B. How WA-LRU works?

We illustrate how WA-LRU works in the following two scenarios. $chunk^A_{index}$ represents the position of client A within the video content, where *index* ranges from 1 to the last chunk of the video content (m).

The two following scenarios (Figures 13 and 13) depict how WA-LRU works:

In Figure 13, client A requests a new chunk which $index$ is superior to Th; If the chunk is not within the cache, then while downloading it from the origin server, the cache will not cache it.

In figure 14, client A and B are requesting the same video content at the same time. Then, if $chunk^A_{index} > chunk^B_{index} +$



(a) Average hit-ratio (b) Average update-ratio (c) Average Hit-ratio per $chunk_{index}$

Fig. 11. *Metrics evaluation: Average hit-ratio, update-ratio and average hit-ratio per chunk position*

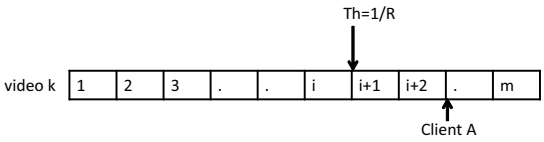


Fig. 13. Scenario 1: comparison between Th and $chunk^A_{index}$

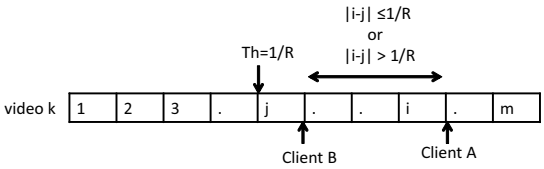


Fig. 14. Scenario 2: comparison between Th and $|chunk^A_{index} - chunk^B_{index}|$

$\Delta_{T=10s}$, at this time, it would be useless to cache $chunk^A_{index}$ since after $10*\Delta_{T=10s}$ seconds, it should be evicted and client B would be redirected to the origin server to download it. If $chunk^A_{index} \leqslant chunk^B_{index} + \Delta_{T=10s}$, then $chunk^A_{index}$ should be kept within the cache since in less than $10*\Delta_{T=10s}$ seconds, client B would reach $chunk^A_{index}$ of the video stream. However $chunk^B_{index}$ should not be cached since the *index* is superior to Th.

C. Pseudo-code of WA-LRU

Let $S_{T=10s} = [s_0, s_1, .., s_N]_{T=10s}$ be the set of active sessions updated each $T = 10s$. s_i could be assimilated to a 3-dimension vector such as $s_i = [Video_{id}, Chunk_{id}, P_k]_i$. In Algorithm 1, we present the pseudo-code of WA-LRU:

- Line 1: updates S_T periodically to capture the active sessions.
- Lines [2-3]: The cache is not full yet and no policy is applied. So we cache all chunks.
- Line 4: The cache is full and chunks are cached and evicted with respect to the WA-LRU policy.
- Lines [5-16]: To decide whether to cache $s_i^{chunk_{id}}$ or not. Scenarios presented in V-B depicts well the mechanism we propose.
- Line 7: By the end of T s, the value R is updated with respect to the number of bytes evicted from the cache.

Algorithm 1 WA-LRU

Require: S_T
1: update(S_T)
2: **if** $R = 0$ **then** ▷
3: $CHUNK :: CACHE$
4: **else**
5: **for** i=N; i=1; i- - **do**
6: **for** j=N-1; j=1; j- - **do**
7: **if** $s_j^{Video_{id}} == s_i^{Video_{id}}$ **then**
8: **if** $s_j^{Chunk_{id}} + \frac{1}{R} \leqslant s_i^{Chunk_{id}}$ **then**
9: $CHUNK :: CACHE$
10: **else**
11: $CHUNK :: DO\,NOT\,CACHE$
12: **end if**
13: **end if**
14: **end for**
15: **end for**
16: **end if**
17: update(R)

D. Evaluation

We evaluate our algorithm along 3 metrics:

- *Average hit-ratio.*
- *Update-ratio*: Ratio of requests leading to update the list of cached chunks.
  Reducing the update-ratio will reduce the amount of processing time within the cache to update the list cached chunks. Thus it would be interesting to reduce this ratio while increasing the hit-ratio.
- *Per-chunk hit ratio*: Average cache hit-ratio per each $chunk_{index}$ of video contents.

We evaluate the performance of the cache for different values of the capacity C such as: $C = \Gamma[\tilde{W}]_{T=10s}$, where $[\tilde{W}]_{T=10s}$ represents the average workload over a period of $T = 10seconds$ measured along the 15 days from the trace-driven simulation. We keep on the assumptions that we introduced in IV-B and we only consider the *multi-profiles* case to fit the real-world characteristics of HAS

We observe in Figure 11(a) that, WA-LRU outperforms LRU when $\Gamma < 50$. This means that when the cache avoids updating blindly the list of cached chunks (especially chunks forming the tail), therefore, we give more chance to the

earlier $chunk_{index}$ to persist more within the cache. On the other hand, we observe in Figure 11(b) that WA-LRU reduces significantly the update-ratio when $\Gamma < 100$. This is due to the mechanism we have presented: chunks that would not be requested would not be cached. As a consequence, this relieves the cache from making unnecessary cache-updates and thus reduces significantly the amount of fetching and processing time within the cache. In Figure 11(c), we show the gain in per-chunk hit-ratio when using WA-LRU against LRU. We show that for low cache sizes, we may achieve at least a gain of 3% for the 3 first requested chunks of video contents. This is important for the $joining - phase$ since clients would be served from the cache rather than from the origin server, and since this is the most critical phase that impacts the *user-engagement* during the rest of the video stream. Moreover we observe that WA-LRU enhances the per-chunk hit ratio for most of the advanced chunks position (at the tail) since they would not be cached, unless if the cache predicts that in the near future they will be requested (*i.e.* case of scenario 2: $chunk^A_{index} \leqslant chunk^B_{index} + \Delta_{T=10s}$).

However, one limitation of our approach is that we can not predict when clients will abort their sessions. For instance, by referring to scenario 2 represented in Figure 14, if clients A and B are watching the same video content, such as: $chunk^A_{index} \leqslant chunk^B_{index} + \Delta_{T=10s}$, then WA-LRU will decide to cache $chunk^A_{index}$ because it considers that client B will request it before its eviction. However, client B may abort his session before reaching it. This makes the prediction not accurate and this explains why in Figure 11(c), when we set $\Gamma = 2$, we observe that the hit-ratio per chunk position might be higher when using LRU rather than WA-LRU. However, this is so infrequent as shown in Figure 11(c), and may happens only for chunks with high indexes.

VI. Related works

HTTP Adaptive Streaming has hugely attracted various TV broadcasters and industries specialized in delivering video contents. In [13], authors analyzed a large-scale Chinese TV service provider. They studied the influence of handled devices on users viewing habits. More specifically, they analyzed channels popularity and compared the access frequencies of mobile TV channels against IPTV systems. In [14], authors provided interesting comparison between the three existing types of streaming: Progressive download, Progressive download with byte range, and HLS. However they did not provide a thorough analysis about the profile-based analysis as we have done. In [15], authors proposed a new caching algorithm that leverages the segmented nature of HAS contents and gives priority to the chunks that would be requested shortly given the latest clients' requests. In [16], authors proposed iDASH to assess the benefits of combining the Scalable Video Coding (SVC) with HAS, and studies the impact on caching efficiency.

All these studies provide complementary insights, yet none of them has been focusing on the profile based analysis as we do here. In this paper, we analyze HAS traffic based on real measurements provided by a large scale European operator. This enables to have a fine-grained study of user behavior and mobile access patterns. We further proposed WA-LRU a new caching algorithm that leverages the observations we have made on clients' behavior.

VII. Conclusion

We believe that these observations can be leveraged to provide guidelines for designing and tuning adequate content delivery systems and mechanisms for mobile networks, including for content caching logic, as well to model clients' behavior in that context. We collected the first *large* and *timely* dataset of its kind that allowed us to characterise the switching pattern and clients' behavior when requesting HTTP adaptive streaming contents. Then we ended the paper by proposing an effective caching algorithm that leverages the time-structure of video chunks within video contents.

References

[1] T. Cisco, "Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update , 2012 – 2017," Tech. Rep., 2013.

[2] A. Gouta, C. Florin Hong, D. Hong, A.-M. Kermarrec, and Y. Lelouedec, "Large scale analysis of http adaptive streaming in mobile networks," in *WoWMoM*. IEEE, 2013.

[3] Q. Shao, D. S. Sharp, N. Cackov, and N. Laskovic, "Analysis of Public Safety Traffic on Trunked Land Mobile Radio Systems," vol. 22, no. 7, pp. 1197–1205, 2004.

[4] C. W. Leong, W. Zhuang, Y. Cheng, and L. Wang, "Call Admission Control for Integrated On / Off Voice and Best-Effort Data Services in Mobile Cellular Communications," vol. 52, no. 5, pp. 778–790, 2004.

[5] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 1–14.

[6] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 157–168.

[7] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the impact of video quality on user engagement," *SIGCOMM-Computer Communication Review*, vol. 41, no. 4, p. 362, 2011.

[8] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 133–144, 2007.

[9] H. Yin, X. Liu, F. Qiu, N. Xia, C. Lin, H. Zhang, V. Sekar, and G. Min, "Inside the bird's nest: measurements of large-scale live vod from the 2008 olympics," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 442–455.

[10] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP – Standards and Design Principles," in *MMSys '11 Proceedings of the second annual ACM conference on Multimedia systems*, 2011, pp. 133–143.

[11] M. Liu, C., Bouazizi, I., & Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," in *ACM MMSys*, 2011, pp. 169–174.

[12] K. J. Ma, "HTTP Live Streaming Bandwidth Management using Intelligent Segment Selection," in *GLOBECOM 2011*, 2011, pp. 1–5.

[13] R. Li, Y., Zhang, Y., & Yuan, "Measurement and Analysis of a Large Scale Commercial Mobile Internet TV System," in *IMC'11*, 2011, pp. 209–224.

[14] J. Erman, A. Gerber, K. K. Ramakrishnan, S. Sen, and O. Spatscheck, "Over The Top Video : the Gorilla in Cellular Networks," in *IMC*, 2011.

[15] D. Hong, D. De Vleeschauwer, F. Baccelli *et al.*, "A chunk-based caching algorithm for streaming video," in *NET-COOP 2010-4th Workshop on Network Control and Optimization*, 2010.

[16] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec, "idash: improved dynamic adaptive streaming over http using scalable video coding," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 257–264.